Comparison Study of Metamodels and Models Co-Evolution Approaches

F. Anguel
Chadli Bendjedid University.
El Tarf, Algeria.
LISCO Laboratory.
Badji Mokhtar University.
Annaba, Algeria
fanguel@yahoo.fr

A. Amirat
Mohammed Chérif Messaadia University.
Souk-Ahras, Algeria
abdelkrim.amirat@yahoo.com

N. .Bounour
LISCO Laboratory.
Badji Mokhtar University.
Annaba, Algeria
nora_bounour@yahoo.fr

***Abstract*— Models have been used in various engineering fields to help managing complexity and represent information in different abstraction levels according to specific notation and stakeholder's viewpoint. Model-Driven-Engineering (MDE) gives basic principles for the use of models as primary artefacts throughout the software development phases. Models are defined using modelling languages defined as metamodels. When a metamodel evolves, models may no longer conform to it. To be able to use these models with the new modelling language, they need to be migrated. In fact, several approaches have been proposed addressing this problem. Some of these approaches tackle the problem by specifying manual solutions. Others either propose matching mechanisms to adapt models or define coupled operator for performing migration. In this paper, we introduce co-evolution problem, we give an overview of different approaches to the problem and compare them. As a complementary result we conclude with directions of future work.**

***Keywords*— Model driven engineering, Model, Metamodel, Co-evolution, Transformation,**

I. INTRODUCTION

Modelling is essential to human activity because every action is preceded by the construction (implicit or explicit) of a model [1]. There are plenty of practical usages of models; particularly in computer science where software models are constructed. Meta-modelling has become the key technology to define domains specific modelling languages for model-driven engineering (MDE). MDE is increasingly emerging as a discipline which strictly prescribes designers to develop software in terms of models rather than programs [2]. According to this perspective, models are leveraged to first-class status. Evolution is unavoidable and affects the whole software lifecycle. Analogously to software, metamodels are subject to evolutionary pressure too [3]. However, changing a metamodel might compromise the related artefacts, whose validity must be restored. In fact modelling languages can change quite frequently which requires the evolution of their metamodels as well as the migration (or adaptation) of their dependent artefacts such as models, editors, interpreters, transformations. Recently, several approaches addressing the problem of co-evolution have been proposed. Mostly, focussing on metamodel and model co-evolution (i.e. model migration). The model migration is a crucial activity and is intrinsically complex and results in a time consuming and error-prone [4] process if no adequate support is provided. Building an automated migration strategy is not trivial and complicated as it has to ensure the preservation of the meaning of a possibly unknown set of models.

In this survey, we discuss the state-of the-art in metamodel and model co-evolution approaches highlighting their strengths and weaknesses, and then we compare a selected set from the described approaches using general criteria that we deem important for model migration. This study allowed us defining some guidelines to develop a novel approach to manage metamodel and model co-evolution.

The remainder of this paper is organised as follows, in section 2 basic concepts related to model and metamodel co-evolution are defined. In section 3 we present an overview of co-evolution approaches in MDE with their categorisation. In section 4 we compare them according to general criteria. In section 5 we define some guidelines in order to develop novel model migration approaches. Finally in section 6 we present our conclusion.

II. MODEL AND METAMODEL CO-EVOLUTION

An MDE system basically consists of metamodels, models, and transformations. A model represents a view of a system and is defined in the language of its metamodel. In other words, a model contains elements conforming to concepts and relationships expressed in its metamodel. A metamodel can be given to define correct models. In the same way a model is described by a metamodel, a metamodel in turn has to be specified in a rigorous manner; this is done by means of meta-metamodels. This may be seen as a minimal definition in support of the basic MDE principle "Everything is a model" [1]. The two core relations associated to this principle are called representation "representedBy" and conformance "conformTo". In this respect, OMG [5] has introduced the four level architecture which organizes artefacts in a hierarchy of model layers (M0, M1, M2, and M3). Models at every level conform to a model belonging to the upper level.

M0 is not part of the modelling world, so the four level architecture should more precisely be named 3+1 architecture [2] as depicted in Fig.1. One of the best-known metamodels in the MDA is the UML metamodel; MOF (Meta-Object Facility) is the metametamodel of OMG that defines UML [5].

Due to changing requirements and technological progress and like other software artefacts, metamodels evolve over time during their life cycle [3]. The addition of new features and/or the resolution of bugs may change metamodels, thus causing possible problems of inconsistency to existing models which conform to the old version of the metamodel and may become not conform to the new version. Therefore to maintain consistency, metamodel evolution requires model adaptation, i.e., model migration, as shown in Fig. 2; so these two steps are referred as model and metamodel co-evolution. Furthermore, model adaptations should be done by means of model transformations. A model transformation takes as input a model conforming to a given metamodel and produces as output another model conforming to a given metamodel.

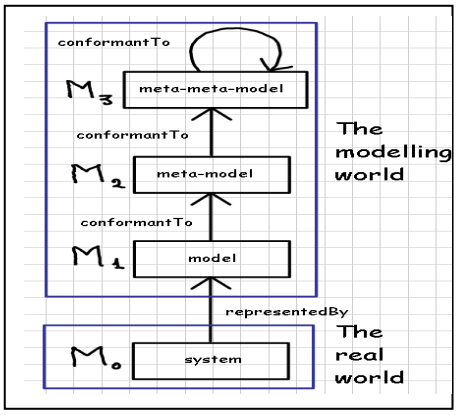

Fig.1. The 3+1 MDA organisation [2]

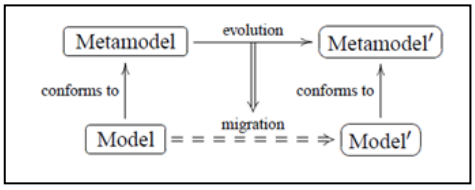

Fig.2. Metamodel and model co-evolution [1].

III. Existing Approaches

Over the last few years, the problem of metamodel evolution and model co-evolution has been investigated by several works [6-13]. Each approach presents strengths and some limits. Firstly we present a categorisation of these approaches and then we provide an overview of each one.

A. Classification of approaches

The Actual Categorisations of evolution and co-evolution approaches focuses either on the information type considered during evolution either on the technique used for migration strategy. Some works classify kinds of model evolution tasks into two categories: syntactic model evolution and semantic model evolution [6]:

- Syntactic model evolution: Basically, this method will modify the existing domain models such that the models obey the syntactic rules of the new language (metamodel). One drawback to a syntactic evolution is that the new data models will not necessarily reflect the intended semantics of the old domain. However, syntactic migration can be fully automatable.
- Semantic model evolution is a transformation or a set of transformations that rewrites a model to have the same meaning in its new language that it had in its original language. Semantic migration requires manual adaptation from the evolver.

Other researches [14], classify model migration approaches into three categories: manual specification, operator-based and matching metamodel approaches.

- In manual specification, the migration strategy is encoded manually by the metamodel developer, typically using a general purpose programming language (e.g. Java) or a model-to-model transformation language (such as QVT [5], or ATL [15]. Approaches classified as manual specification are essentially Sprinkle approach [6], Narayanan approach [6] and Rose approach [7].
- Operator-based approaches specify metamodel evolution by a sequence of operator applications. Each operator application can be coupled to a corresponding model migration strategy. In these approaches a library of co-evolutionary operators is provided. By composing co-evolutionary operators, metamodel evolution can be performed and a migration strategy can be generated without writing any code. The significant approaches of this category are Wachsmuth approach [12] and Herrmannsdoerfer approach [13].
- In metamodel matching, a migration strategy is inferred by analyzing the evolved metamodel and the metamodel history [8]. Metamodel matching approaches use one of two categories of metamodel history; either the original metamodel (differencing approaches) or the changes made to the original metamodel to produce the evolved metamodel (change recording approaches).In this category we find several approaches like Gruschko approach [9], Cicchetti approach [10] and Garcés approach [11].

B. Approaches Overview

1) A domain specific visual language (DSVL) for domain model evolution: Sprinkle's approach [6],[16] defines a domain specific visual language (DSVL) developed expressly for the evolution of domain specific visual languages. It provides an interface that is specialized for describing an algorithm to transform domain models from one DSVL to another. The migration of domain models is performed by using syntactic patterns of domain concepts that are mapped to patterns of evolved domain concepts through mapping rules; these rules follow a "pattern implies consequence" form. The mapping are associations between pattern and consequences, or attributes of a pattern or consequence, and are formed from a fundamental set of operations such as "Create", "Create within", "Becomes" and "Delete". The Transformation is made up of sequenced Transforms that are used to describe the

specific differences between metamodels. Each transform will generate an XSL document. The model of computation for sequenced transforms operates on the input domain models with the first transform and continues on through until there are no further transforms to apply. In this approach, if the user is familiar with metamodeling concepts with very little guidance that can create a domain evolution transformation that will evolve the domain models in the evolved DSVL [16]. The powerful of the provided language is creating syntax patterns in any form, since the pattern language is derived from the metamodeling language. However efficiency of the conceived transformation algorithm depends on the ability of the modeler. We note also that the language is unable to check for the correctness of the transformed domain models [6].

2) Automatic Domain Model Migration to Manage Metamodel evolution: Narayanan's approach defines MCL language "Model change language" [7] using a MOF-compliant metamodel. MCL is a high-level visual language for describing metamodel evolution. MCL defines a set of idioms and a composition approach for specification of the migration rules. Rules can be used to specify most of the common metamodel evolution cases (e.g. adding new concept, modifying an element, deleting an element, adding new subtypes and modifying local, and automate the migration of instance models. MCL used basic pattern for typical migration scenarios that consists of an LHS element from the old metamodel, an RHS element from the new metamodel, and a "MapsTo" relation between them. Another special link, called the "WasMappedTo" link, in the pattern is used to match a node that was previously migrated, by an earlier migration rule. As opposed to providing a general transformation interface for the migrator like in sprinkle approach [16], MCL provides a Domain specific modeling language DSML as the specification language, so MCL is more efficient. We note that MCL provide straightforward graphical syntax and semantics is rather simple, MCL is modular, expressive and allow reusing of knowledge migration. MCL can also specify complex relations between meta-entities. But, in MCL some rules must be resolved manually and there are cases that depend on the intention of the transformation developer.

3) *Model Migration with Epsilon Flock:* In Rose approach, Flock is a domain-specific language for specifying and executing model migration strategies [8]. Flock uses a model connectivity framework, which decouples migration from the representation. Flock has a compact syntax. Much of its design and implementation is focused on the runtime. Flock automatically maps each element of the original model to an equivalent element of the migrated model using a novel conservative copying algorithm and user-defined migration rules, when original model elements conform to the evolved metamodel The conservative copy algorithm copies model elements from original to migrated model. Hence the user specifies migration only for model elements which no longer conform to the evolved metamodel. Flock migration strategies are organized into modules, which inherit from EOL modules. Modules comprise any number of rules. Flock delegate conformance checking responsibilities to EMC. It seems clear that approach gets its strength from the connectivity layer of epsilon that allow Flock to use models represented in MDR, XML, and CZT and Flock is able to be extended to support further modelling technologies. However encoding migration strategy becomes more difficult for larger metamodels since there is no tool support for analyzing the changes between original and evolved metamodel.

4) *Towards synchronizing models with evolving metamodels:* Gruschko's approach [9] is model-to-model transformation. Envisioned steps of the proposed model migration approach are: firstly model versions are compared and the differences are translated into the delta model. The found changes are classified into categories. Then the user input needed for not automatically resolvable changed migration is gathered. Finally an appropriate algorithm for model migration has to be determined, and the migration is executed. The proposed approach minimizes the manual effort required to perform model migration in face of metamodel changes. But, the changes are assumed to occur individually, and using relations instead of difference models does not allow distinguishing meta-element updates from deletion/addition patterns.

5) *Transformational approach to model co-evolution:* Cicchetti's approach [10] is a co-adaptation approach given as a higher-order model transformation which takes the difference model recording the metamodel evolution and generates a model transformation able to produce the co-evolution of models. The approach consists of the following steps: firstly automatic decomposition of the difference model (□) in two disjoint (sub) models, which denote breaking resolvable (□R) and unresolvable changes (□-R); if (□R) and (□-R) are parallel independent then the corresponding co-evolutions are generated separately, however if (□R) and (□-R) are parallel dependent, they are further refined to identify and isolate the interdependencies causing the interferences. This approach is implemented and available for download [17]. This approach does not specify explicitly how the difference models are calculated, only that they can be obtained by using a tool such as EMFCompare or SiDiff. We note that isolation of the interdependencies between changes is not always possible.

6) *Managing Model Adaptation by precise Detection of Metamodel Changes:* Garcés approach [11] consists of a three-step adaptation in order to adopt models to their evolving metamodels and thus follow a matching approach to co-evolution. Firstly a matching process computes automatically the equivalences and differences between two metamodels versions by incrementally executing a set of heuristics. The computed equivalences and differences are saved in a matching model. Secondly an adaptation transformation is derived by a higher-order transformation tacking as input the matching model. The produced transformation is written in a particular transformation language (e.g. ATL, XSLT, SQL-like). This transformation preserves unchanged model elements and migrate changed ones. Finally, the adaptation transformation is executed. Authors prove that the proposed approach achieves a high accuracy in detecting simple and complex changes and a good performance of matching strategies is also proven. the family of heuristics to design the constructs of the AtlanMod

Matching Language (AML), a Domain-Specific Language (DSL) for expressing matching strategies [18]. We find that this approach is powerful, because it allows computing equivalences and differences between any pair of metamodels, and matching step executes modularized heuristics that may be plugged or unplugged on demand. The approach is generic, heuristics are described in terms of KM3 concepts and it can be implemented using other formalisms such as MOF or EMFEcore. However, user assistance is required in some strategies, and we note that semantically invalid combination of heuristics can cause a runtime error, while an incorrect combination results in the generation of an incorrect migration transformation. Using heuristics is also ambiguous.

7) *Metamodel Adaptation and Model Co-Adaptation:* Wachsmuth's approach is a transformational approach to assist metamodel evolution by stepwise adaptation [12]. The steps are implemented as transformations in QVT Relations. Each step forms a metamodel adaptation. Transformation is classified according to its semantics and instance preservation properties in three groups, namely refactoring, construction, and destruction. This approach is characterized by the possibility to reuse adaptation scripts in similar adaptation scenario and it prevents inconsistencies. But, we find that this approach is very limited because of the atomicity of the changes, which is far from being realist.

8) COPE-automating coupled evolution of metamodels and models: Herrmannsdoefer's approach [13] records the evolution as a sequence of coupled operations in an explicit history model. Each coupled operation encapsulates both metamodel adaptation as well as reconciling model migration. Existing models can be automatically migrated to the adapted version of the metamodel. When no co-evolutionary operator is appropriate COPE allows metamodel developers to specify custom migration strategies, using a general purpose programming language. COPE provides additional tool support to inspect, refactor and recover the coupled evolution. This approach facilitates metamodel analysis, it offers a greater degree of reusing recurring coupled operations in model migration, and it uses large libraries of co-evolutionary operators [19] and improves their navigability by making clearer communication of operators. We note also that COPE is open source. However, determining which sequence of operations will produce a correct migration is not always straightforward specifically for large metamodels. The custom migration in COPE is specified in general purpose programming language which differs from migration strategy language.

IV. COMPARATIVE ANALYSIS

In this section we present a comparative analysis of model and metamodel co-evolution approaches. Approaches described are compared with respect of the general criteria that represent features of these approaches: specification of the evolution, evolution source, migration target, migration language and migration extensibility. The selected criteria are general and could be used to evaluate any co-evolution approach in MDE. Other evaluation criteria, such as performance and conciseness are also feasible to evaluate model migration. Therefore, due to our aim which is identifying requirements for co-evolution approach, we decided to explore in this analysis only these general criteria.

A. *Criteria of comparative Analysis*

1) *Evolution Specification:* The evolution of a metamodel is implicitly specified by the original and the evolved version of the metamodel or it is specified explicitly. Many approaches are based on explicit evolution specifications. We distinguish two styles of such specifications [20]: Imperative specifications describe the evolution by a sequence of applications of change operations. In contrast, declarative specifications model the evolution by a set of differences between the original and evolved version of a metamodel. The specification criterion indicates if the evolution is imperative, declarative or implicit.

2) *Evolution source:* Explicit evolution specifications can have different sources. So, the evolution can be user-defined where the user specifies the evolution manually. Another way is recording evolution while the user edits metamodel changes. The prominent source evolution is the automated detection of the evolution based on the original and evolved version of a definition. We distinguish two kinds of detections: First, detections which are only able to detect simple changes like additions and deletions. For some approaches, this includes the detection of moves as well. Second, detections which can also detect more complex changes, for example extracting and in-lining of constructs.

3) *Migration Target*: Migration might be performed either in-place or out-of-place. In the first case, the target of the migration is the original model itself which is modified during migration. In the second case, the target is a new migrated model which is created during migration. The original model is preserved.

4) *Migration Language:* Migration might be custom defined as a domain specific migration language. Alternatively, an existing transformation language (TL) can be reused. Another way is to add migration support to a general-purpose programming language (GPL) in form of an API or an embedded domain specific language.

5) Migration Extensibility: This criterion defines if extensions are supported by the studied approaches. Three kinds of extensibility can be supported. The fixed migration can be completely defined by the developer and only the developer can add new parts in the migration strategy. Another kind of migration is the over-writable strategy where the user can overwrite and customize single applications of a migration. The third kind consists of an extendable migration where the user can add completely new parts in the migration strategy.

B. *Co-Evolution Approaches Comparison*

The table 1 lists approaches presented in section 3 and shows their comportment relatively to studied criteria.

Manual specification approaches Sprinkle, Narayanan and Rose provide custom model transformation languages to manually specify the model migration. Which reduce the effort for building a migration specification For instance; migrations automatically copy model elements whose metamodel definition has not changed [8].

TABLE 1. COMPARISON OF CO-EVOLUTION APPROACHES.

Approach	Evolution		Migration		
	specification	source	Target	Language	Extensibility
Sprinkle	Declarative	User-defined	Out	custom	Over-writable
Narayanan	Declarative	User-defined	Out	custom	Over-writable
Rose	Implicit	-	Out	custom	Over-writable
Gruschko	Declarative	Detected-simple	Out	TL/ETL	Over-writable
Cicchetti	Declarative	Detected-complex	Out	TL/ATL	Fixed
Garcès	Imperative	Detected-complex	Out	TL/ATL	Extendable
Wachsmuth	Imperative	User-defined	Out	TL/QVT	Fixed
Herremmandoerfer	Imperative	recorded	in	TL/Groovy	Extendable

The user then overwrites this default behavior with the intended migration. Visual languages introduced by Sprinkle [6], [16] and Narayanan [7] specify the differences between two versions of a GME-based and defines a model migration based on these differences. Migration algorithms not covered by MCL can be specified imperatively using a C++ API. Flock is a textual migration language for EMF-based models [8].

Manual specification approaches Sprinkle, Narayanan and Rose provide custom model transformation languages to manually specify the model migration. Which reduce the effort for building a migration specification For instance; migrations automatically copy model elements whose metamodel definition has not changed [8]. The user manual specification approaches Sprinkle, Narayanan and Rose provide custom model transformation languages to manually specify the model migration. Which reduce the effort for building a migration specification For instance; migrations automatically copy model elements whose metamodel definition has not changed [8]. The user then overwrites this default behavior with the intended migration. Visual languages introduced by Sprinkle [6], [16] and Narayanan [7] specify the differences between two versions of a GME-based and defines a model migration based on these differences. Migration algorithms not covered by MCL can be specified imperatively using a C++ API. Flock is a textual migration language for EMF-based models [8].

Here, only the model migration is specified. Differences between metamodel versions are not made explicit. Instead, Flock automatically copies only those model elements which conform to the evolved metamodel. The user then iteratively redefines the migration specification to migrate non-conforming elements. Manual specification approaches do not provide a construct for the reuse of migration knowledge across metamodels [20]. The unique feature of manual specification approaches is a custom migration language for overwriting a default migration manually. This feature increases the expressivity of these approaches.

Metamodel Matching approaches automatically detect the differences between two metamodel versions. These are stored in a declarative difference model from which a migration specification is generated. Gruschko approach support the automatic detection of simple changes in Ecore metamodels [9]. However, Cicchetti et al. also detect complex changes [10], [17] the migration specification consists of a set of model transformations to be executed consecutively. Since this is prevented by interdependent changes. AML (Atlas Matching Language) allows the user to parameterize the detection of complex changes [11]. Therefore, the user combines existing or user-defined heuristics to a matching algorithm. From a difference model obtained by such an algorithm, an ATL transformation specifying the migration is automatically generated. For matching approaches, the unique feature is a declarative evolution specification which is either recorded or detected. This feature permits increasing automaticity by automatic generating migration strategy.

Operation-based approaches provide a set of reusable coupled operations that work at the metamodel level as well as at the model level. Wachsmuth presents an operation suite for the MOF metamodeling formalism, operations are classified according to language and model preservation properties [12] for migration, and the evolution specification is translated into a QVT Relations model transformation. This approach is not expressive enough to capture all kinds of migration scenarios, due to the restricted set of high-level primitives [20]. COPE [13] is a model migration tool that records the metamodel adaptation as a sequence of operations in a history model. COPE uses an imperative language and its migrating transformations are executed in-place. Reuse of recurring migration specifications through parameters and constraints restricting allows reducing the effort associated with building a model migration [22]. Migration specifications can become so specific to a certain metamodel that reuse makes no sense [22]. To express these complex migrations, COPE allows the user to define a custom coupled operation by manually encoding a model migration The unique feature of operation-based approaches is an imperative evolution specification as a sequence of operation applications. This feature favourites the reuse which is a mean to reduce effort. Most approaches perform out-of-place migration. Model transformation languages generally do not support in-place transformations with different source and target metamodels. However, COPE [13] supports in-place migrations with a new transformation language.

V. REQUIREMENTS

After analysing existing approaches and refereeing to other works comparing migration approaches [21-22], we define the following requirements of model and metamodel co-evolution approach:

- The first one is increasing automaticity of model and metamodel co-evolution in response to metamodel evolution as far as possible. For that, It seems useful to combine reusing feature of recurred operations as in operator based approach and copying feature of unchanged elements like in manual specification approaches, and also introducing matching techniques to improve co-evolution process [24].
- The second requirement is increasing clarity and understandability of migration strategy by using matured language of migration [22]. For example, It is benefit to use the Eclipse Modelling Framework (EMF) [23] because it is a well-known and widely used technology and as transformation language ATL [15]. The use of standards tools allows interoperability with other systems.
- The third requirement is increasing expressivity of migration strategy by assuring user driven solution in one hand and permitting extensibility in the other hand.
- The fourth requirement is to assure the flexibility approach by using either recorded evolution or detected changes if only the two versions of metamodel are given.

VI. CONCLUSIONS

The main purpose of this paper is to put the light on metamodel and model co-evolution. Firstly we have study the existing approaches that treat model migration in response to metamodel evolution. We have seen different classification of these approaches. Then we have selected some criteria that we consider significant to evaluate the studied approaches, after the comparison analysis process we find that no approach cover the overall criteria.

Therefore driven by this analysis, we have defined guidelines to solve co-evolution problem with more expressivity and clarity and supporting change and extensibility of migration strategy to ensure its correctness. Furthermore, using standard tools like EMF and ATL allow large diffusion of the solution and facilitate its interoperability with other systems.

REFERENCES

[1] J., Bézivin, 2005. On the Unification Power of Models. Software and Systems Modeling (SoSyM.), vol. 4(2), pp. 171–188.

[2] Bézivin, J.: In search of a basic principle for model driven engineering. UPGRADE Eur. J. Inf. Prof. 5(2), 21–24 (2004)

[3] J.M., Favre, 2003. Meta-model and model co-evolution within the 3D software space. In Proc. ELISA'03 Workshop. pp 98–109.

[4] M., Herrmannsdoerfer, S., Benz and E., Juergens. 2008. Automatability of Coupled Evolution of Metamodels and Models in Practice" in Proc. MoDELS'08. LNCS Springer, vol. 5301, pp. 645-659.

[5] OMG, "MOF QVT Final Adopted Specification," Available: www.omg.org/docs/ptc/05-11-01.pdf , 2005.

[6] J. Sprinkle, 2003. Metamodel driven model migration. Phd. thesis, Vanderbilt University.

[7] A. Narayanan, T. Levendovszky, D. Balasubramanian and G. Karsai, "Automatic domain model migration to manage metamodel evolution," in Proc. MODELS'09, 2009, LNCS Springer, vol. 5795, pp. 706-711.

[8] L.M. Rose, D.S.Kolovos, R.F.Paige and F.A.C. Polack, 2010. Model migration with Epsilon Flock. In Proc ICMT'10, 2010, LNCS Springer, vol. 6142, pp. 184-198.

[9] B., Gruschko, D.S., Kolovos and R.F, Paige, 2007. Towards synchronizing models with evolving metamodels. In Proc. the International Workshop on Model-Driven Software Evolution.

[10] A., Cicchetti, D.Di., Ruscio, R., Eramo and A., Pierantonio, 2008. Automating co-evolution in MDE. In Proc. EDOC'08 IEEE Computer Society. pp 222-231.

[11] K., Garcès, F., Jouault, P., Cointe and J., Bézivin, 2009. Managing Model Adaptation by Precise Detection of Metamodel Changes. In Proc ECMDA-FA'09. LNCS Springer, vol. 5562. pp 34-49.

[12] G. Wachsmuth, 2007. Metamodel adaptation and model co-adaptation. In Proc. ECOOP'07.LNCS Springer, vol. 4609, pp. 600-624.

[13] M., Herrmannsdoerfer, S., Benz, and E.Juergens, 2009. COPE – automating coupled evolution of metamodels and models. In Proc. ECOOP09. LNCS Springer, vol. 5653, pp. 52-76.

[14] L.M. Rose, D.S. Kolovos, R.F. Paige, and F.A.C. Polack, 2009. An analysis of approaches to model migration. In Proc. Joint MoDSE-MCCM Workshop,.

[15] F. Jouault and I. Kurtev, 2005. Transforming models with ATL. In Proc. Satellite Events at MoDELS. LNCS Springer, vol. 3844, pp. 128-138.

[16] J. Sprinkle and G. Karsai, 2004. A domain-specific visual language for domain model evolution. Journal of Visual Languages and Computing. vol. 15, pp. 291-307.

[17] A., Cicchetti, 2008. Difference Representation and Conflict Management in Model-Driven Engineering. Phd. Thesis, Computer science Dept University of L'Aquila.

[18] K., Garcès, F., Jouault, P., Cointe and J., Bézivin, 2009. A Domain Specific Language for Expressing Model Matching. In Proc. IDM09.

[19] M. Herrmannsdoerfer, S. D. Vermolen, and G. Wachsmuth, "An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models," In SLE'10: LNCS, vol. 6563, pp. 163–182. Springer,Berlin 2011.

[20] M.Herrmannsdoerfer, "COPE – A Workbench for the coupled evolution of metamodels and models in Proc. SLE'10, 2010, pp. 286-295.

[21] L.M. Rose, D.S.Kolovos, R.F.Paige and F.A.C. Polack, 2010. Comparaison of Model migration Tools,. In Proc. MODELS'10. LNCS Springer, vol. 6394,pp. 61–75.

[22] L.M. Rose, M. Herrmannsdoerfer, S. Mazanek, P.V. Gorp, S. Buchwald, T. Horn, E. Kalnina, A.Koch, K. Lano, B. Schätz, M. Wimmer, "Graph and model transformation tools for model migration," Software and System Modelling, 2012.

[23] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks,: "EMF: Eclipse Modeling Framework 2.0," Addison-Wesley , 2009.

[24] M.Herrmannsdörfer, G. Wachsmuth: "Coupled Evolution of Software Metamodels and Models". Book chapter pp 33-63. In "Evolving Software Systems". Mens, Tom, Serebrenik Alexander, Cleve, Anthony (Eds.), 404 p, Springer, 2014.